CSE 590
Data Science Fundamentals

Pattern And Association Mining

Klaus Mueller

Computer Science Department
Stony Brook University and SUNY Korea

Lecture	Topic	Projects
1	Intro, schedule, and logistics	
2	Data Science components and tasks	
3	Data types	Project #1 out
4	Introduction to R, statistics foundations	
5	Introduction to D3, visual analytics	
6	Data preparation and reduction	
7	Data preparation and reduction	Project #1 due
8	Similarity and distances	Project #2 out
9	Similarity and distances	
10	Cluster analysis	
11	Cluster analysis	
12	Pattern mining	Project #2 due
13	Pattern mining	
14	Outlier analysis	
15	Outlier analysis	Final Project proposal due
16	Classifiers	
17	Midterm	
18	Classifiers	
19	Optimization and model fitting	
20	Optimization and model fitting	
21	Causal modeling	
22	Streaming data	Final Project preliminary report due
23	Text data	
24	Time series data	
25	Graph data	
26	Scalability and data engineering	
27	Data journalism	
	Final project presentation	Final Project slides and final report due

Frequent Pattern (FP) Analysis

Frequent pattern:

- a pattern (set of items) that occurs frequently in a data set
- called frequent itemsets (Agrawal et al. (1993)

Motivation: find inherent regularities in data

- what products were often purchased together?
- the classic example: beer and diapers?
- what are the subsequent purchases after buying a PC?
- what kinds of DNA are sensitive to this new drug?
- can we automatically classify web documents?

Applications

- basket data analysis, cross-marketing, catalog design, sales campaign analysis, Web log (click stream) analysis, DNA sequence analysis

Transaction data –supermarket

Market basket transactions:

t1: {beer, nuts, diaper}

t2: {beer, coffee, diaper}

… …

tn: {nuts, coffee, diaper., eggs, milk}

Concepts:

- an *item*: an item/article in a basket
- *I*: the set of all items sold in the store
- A *transaction*: items purchased in a basket; it may have a TID (transaction ID)
- A *transactional dataset*: A set of transactions

Importance of FP Mining

Frequent pattern:

- an intrinsic and important property of datasets

Foundation for many essential data mining tasks

- association, correlation, and causality analysis
- sequential, structural (e.g., sub-graph) patterns
- pattern analysis in spatiotemporal, multimedia, time-series, and stream data
- classification: discriminative, frequent pattern analysis
- cluster analysis: frequent pattern-based clustering
- data warehousing
- semantic data compression
- other broad applications

Frequent Patterns

itemset: a set of one or more items

- X = {milk, bread, cereal} is an itemset

k-itemset: $X = \{x_1, \ldots, x_k\}$

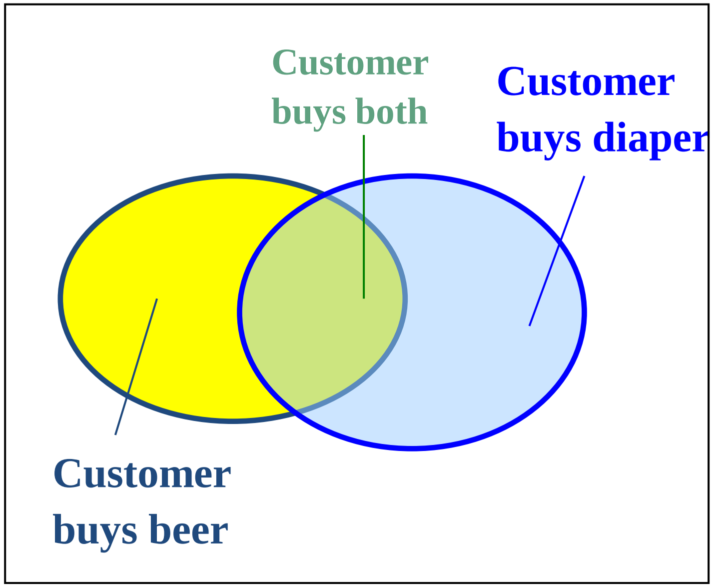
- {milk, bread, cereal} is a 3-itemset

(absolute) support, or, *support count* of X: frequency or occurrence of an itemset X

(relative) support, *s*, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)

an itemset X is *frequent* if X's support is no less than a *minsup* threshold

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

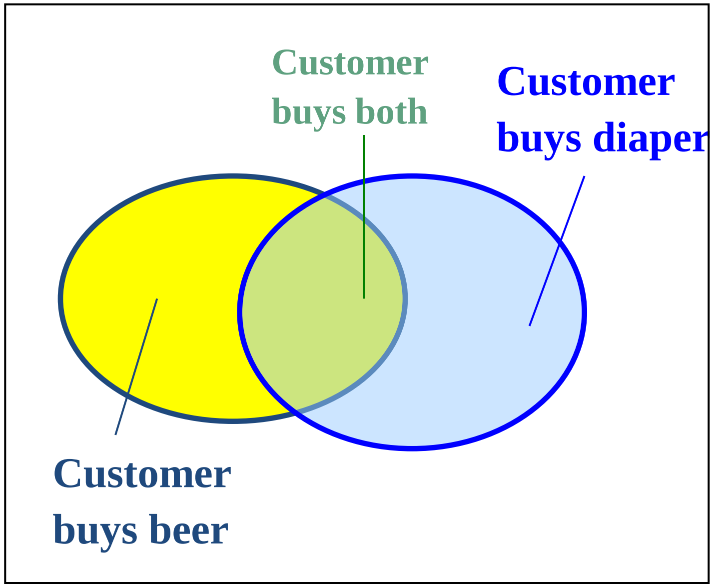


Association Rules

Find all the rules X → Y with minimum support and confidence

- support, s, probability that a transaction contains X ∪ Y
- confidence, c, conditional probability that a transaction having X also contains *Y*

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



Example:

- let minsup=50%, minconf=50%
- FP: Beer: s=3/5, Nuts: s=3/5, Diaper: s=4/5, Eggs: s=3/5, {Beer, Diaper}: s=3/5

Association rules: (many more!)

- Beer → Diaper
  (s(B)=60%, c(D|B)=100%)
- Diaper → Beer
  (s(D)=80%, c(B|D=75%)

Pattern Mining Algorithms

Realistic databases have a large number of patterns

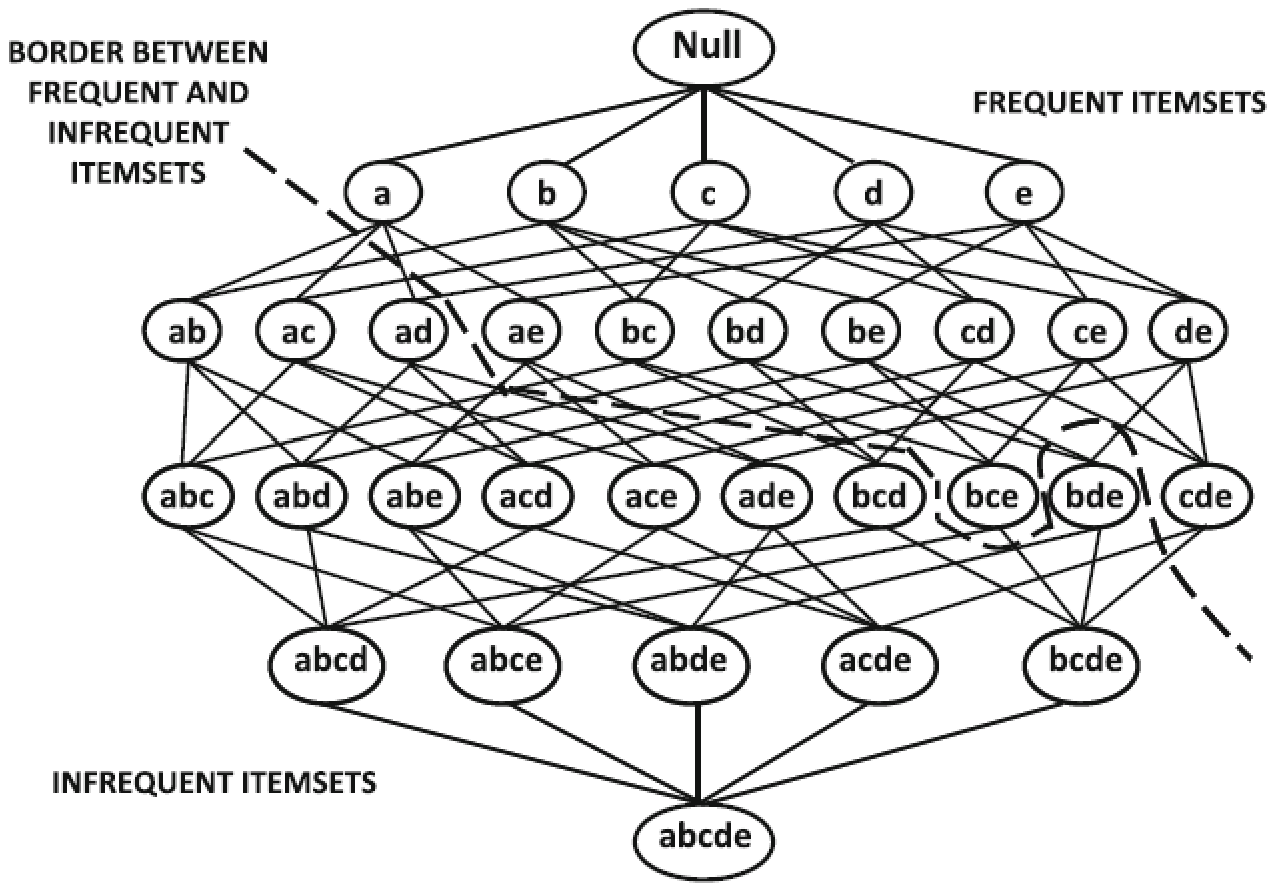
- can be expensive to mine
- hence there are many mining algorithms
- they use different strategies and data structures
- their resulting sets of rules are all the same.

Given a transaction data set T, and a minimum support and a minimum confidence, the set of association rules existing in T is uniquely determined

- any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different

The Itemset Lattice

Lexicographically order the items



Differentiation of Mining Algorithms

Differ in how they grow the lexicographic or *enumeration tree* of frequent itemsets

- trade-offs between storage, disk access costs, comp. efficiency
- breadth first
- depth first

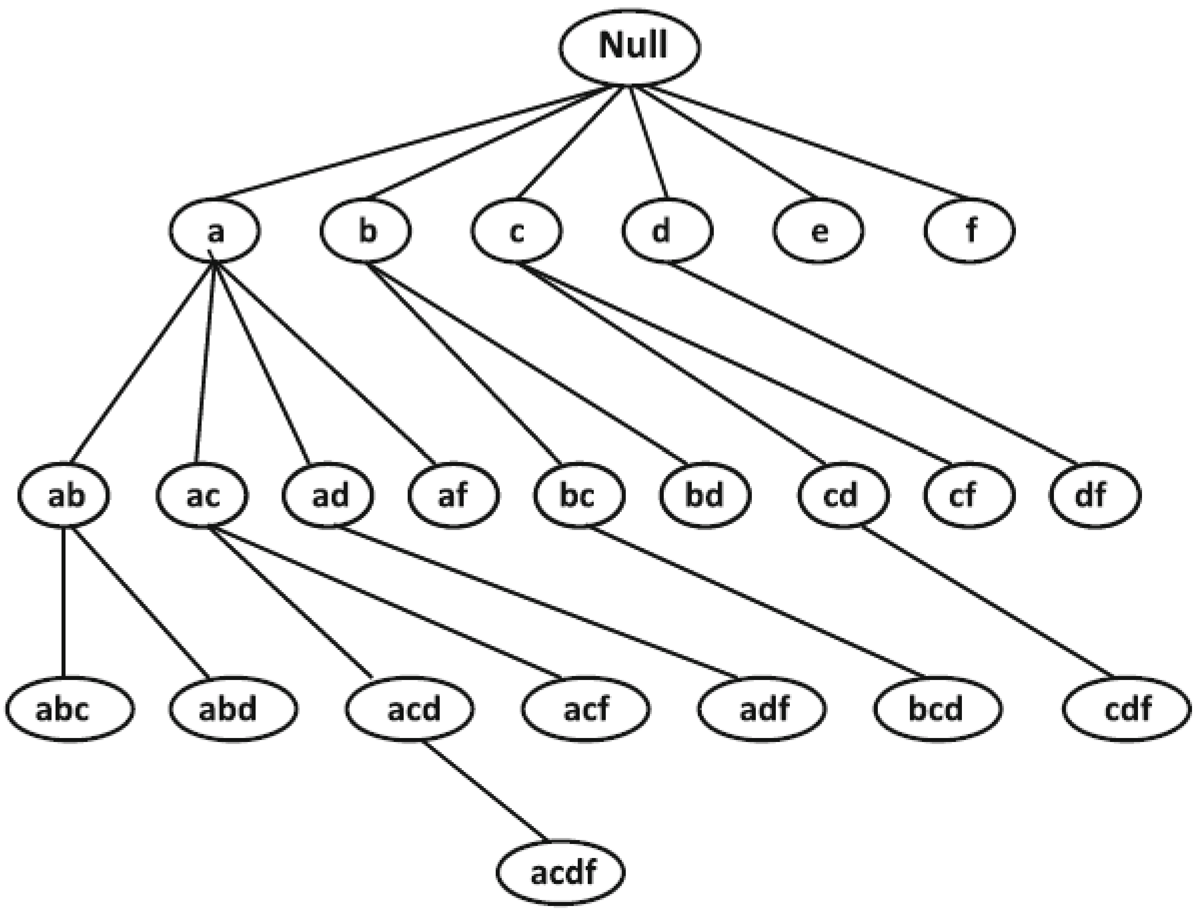
Breadth-first

- more relevant for disk-resident databases
- all nodes at a single level of the tree can be extended during one counting pass on the transaction database

Depth-first

- better ability to explore the tree deeply and discover long frequent patterns early
- useful to gain computational efficiency in maximal pattern mining

ENUMERATION TREE OF FREQUENT ITEMSETS



Generic enumeration-tree growth Algorithm

Unspecified growth strategy and counting method

Algorithm *GenericEnumerationTree*(Transactions: $\mathcal{T}$,
Minimum Support: *minsup*)
begin
  Initialize enumeration tree $\mathcal{ET}$ to single $Null$ node;
  while any node in $\mathcal{ET}$ has not been examined **do begin**
    Select one of more unexamined nodes $\mathcal{P}$ from $\mathcal{ET}$ for examination;
    Generate candidates extensions $C(P)$ of each node $P \in \mathcal{P}$;
    Determine frequent extensions $F(P) \subseteq C(P)$ for each $P \in \mathcal{P}$ with support counting;
    Extend each node $P \in \mathcal{P}$ in $\mathcal{ET}$ with its frequent extensions in $F(P)$;
  end
  return enumeration tree $\mathcal{ET}$;
end

Algorithms for Tree Growth

Many algorithms

- projection-based
- recursive
- hash-table assisted
- optimized counting at deeper level nodes
- pointer-less, array-based trees

We shall look at the Apriori algorithm in more detail

- many more are in the text book

The Downward Closure Property

The downward closure property of frequent patterns

- any subset of a frequent itemset must be frequent
- if {beer, diaper, nuts} is frequent, so is {beer, diaper}
- i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- this can be useful to prune unnecessary searches
- this leads to the Apriori pruning principle

Apriori pruning principle:

- if there is any itemset which is infrequent, then its superset should not be generated/tested

Apriori Algorithm

Pioneered by

- Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD' 94

Method:

- initially, scan DB once to get frequent 1-itemsets
- generate length (k+1) candidate itemsets from length k frequent itemsets
- test the candidates against DB
- terminate when no frequent or candidate set can be generated

Apriori Algorithm – Example

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1^{st} scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2^{nd} scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3^{rd} scan

L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm – Pseudo Code

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
    increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
return $\cup_k L_k$;

Implementation of Apriori

How to generate candidates?

- Step 1: self-joining L_k
- Step 2: pruning

Example of Candidate-generation

- L_3={*abc, abd, acd, ace, bcd*}
- Self-joining: L_3*L_3
  - *abcd* from *abc* and *abd*
  - *acde* from *acd* and *ace*
- Pruning:
  - *acde* is removed because *ade* is not in L_3
- C_4 = {*abcd*}
- can also limited the depth, k, of the tree for efficiency

Alternative Models: Interesting Patterns

Why Use Alternative Algorithms?

Frequent itemset generation has found widespread popularity and acceptance

- simple
- downward closure property very helpful for pruning

However,

- patterns found are NOT always significant from an *application-specific* perspective.
- raw frequencies of itemsets do not always correspond to the most *interesting* patterns

Example of Shortcomings

Database in which *all* the transactions contain the item *Milk*

- therefore, the item *Milk* can be appended to *any* set of items, without changing its frequency.
- however, this does not mean that *Milk* is truly associated with any set of items

In this case for any set of items X, the association rule $X \Rightarrow \{Milk\}$ has 100% confidence

- however, it would not make sense for the supermarket merchant to assume that the basket of items X is *discriminatively* indicative of *Milk*
- this is the limitation of the traditional support-confidence model

- *for example, "Buy walnuts* $\Rightarrow$ *buy milk* [1%, 80%]" is misleading if 85% of customers buy milk

INTERESTINGNESS-BASED MODELS

It is possible to quantify the affinity of sets of items in ways that are statistically more robust than the support-confidence framework

However, the major computational problem is that the downward closure property is generally not satisfied

This makes algorithmic development rather difficult on the exponentially large search space of patterns

In some cases, the measure is defined only for the special case of 2-itemsets

CORRELATION

Pearson's coefficient:

$$\rho = \frac{E[X \cdot Y] - E[X] \cdot E[Y]}{\sigma(X) \cdot \sigma(Y)}$$

Adapted to the notion of relative support:

$$\rho_{ij} = \frac{sup(\{i,j\}) - sup(i) \cdot sup(j)}{\sqrt{sup(i) \cdot sup(j) \cdot (1 - sup(i)) \cdot (1 - sup(j))}}.$$

Why does it better with regards to the milk example just quoted?

χ2 Measure

For a set of *k* binary random variables (items), denoted by *X*, there are 2^k-possible states representing presence or absence of different items of *X* in the transaction

For example, for *k* = 2 items *{Bread, Butter*

- there are 2^2 states
- *{Bread,Butter}*, *{Bread, ¬Butter}*, *{ ¬Bread,Butter}*, *{ ¬Bread, ¬Butter}*
- their expected fractional presences = as the product of the supports of the states (presence or absence) of the individual items

χ2 compares these expected states with the observed:

$$\chi^2(X) = \sum_{i=1}^{2^{|X|}} \frac{(O_i - E_i)^2}{E_i}$$

$\chi 2$ Measure – Example

When *X* = *{Bread,Butter}*

- perform the summation over the $2^2 = 4$ states corresponding to *{Bread,Butter}*, *{Bread, ¬Butter}*, *{ ¬Bread,Butter}*, *{ ¬Bread, ¬Butter}*.
- a value that is close to 0 indicates statistical independence among the items → no relation in behavior
- larger values of this quantity indicate greater dependence between the variables → there is a relation in behavior

However,

- large $\chi 2$ values do not reveal whether the dependence between items is positive or negative
- this is because the $\chi 2$ test measures dependence between variables, rather than the nature of the correlation between the specific states of these variables
- can look at the $\chi 2$ table to see for what of the states O is different from E

$\chi 2$ Measure – Implementation

The $\chi 2$-test satisfies the *upward closure property*

- this enables an efficient algorithm discovering interesting *k*-patterns
- however, the computational complexity increases exponentially with $|X|$

Other interestingness Metrics

See text book

- cosine coefficient
- interest ratio
- Jaccard coefficient
- collective strength

Slide Credits

Content of some slides courtesy of

- Jiawei Han, Micheline Kamber, and Jian Pei, UIUC
- Bing Liu, UIC